1-1-1988

An Exact Algorithm for Variations of the Assembly line Balancing Problem

Jay E. Aronson
The University of Georgia

Gary Klein
Southern Methodist University

AN EXACT ALGORITHM FOR VARIATIONS OF THE ASSEMBLY LINE BALANCING PROBLEM

Working Paper 88-062*

by

Jay E. Aronson
Gary Klein

Jay E. Aronson
The University of Georgia

Gary Klein
Assistant Professor
Management Information Sciences
Edwin L. Cox School of Business
Southern Methodist University
Dallas, Texas 75275

* This paper represents a draft of work in progress by the authors and is being sent to you for information and review. Responsibility for the contents rests solely with the authors and may not be reproduced or distributed without their written consent. Please address all correspondence to Gary Klein.

AN EXACT ALGORITHM FOR VARIATIONS OF THE ASSEMBLY LINE BALANCING PROBLEM

Jay E. Aronson

Gary Klein

ABSTRACT

We discuss a general formulation for the assembly line balancing problem. The formulation allows for the common extensions plus the inclusion of an objective function that minimizes costs associated with placing conflicting tasks within the same station. We also discuss a variation with a fixed-charge of using a station. We provide a general solution algorithm that extends existing methodologies for solving the simple assembly line balancing problem to the solution of the more general formulations. We discuss implementation issues and present computational results for sets of assembly line balancing problems described in the literature.

KEY WORDS: 583 PRODUCTION/SCHEDULING - FLOW SHOP;
630 PROGRAMMING - INTEGER ALGORITHMS, ENUMERATIVE;
482 NETWORKS/GRAPHS - APPLICATIONS.

1. Introduction

An assembly line performs a set of identifiable tasks grouped into a series of independent stations. The tasks are processed entirely at individual stations and have a predefined set of precedence requirements that restrict their assignment to stations. The stations are arranged in a sequential order and all tasks assigned to the station are completed prior to sending the item along the line to the next station in the series of stations. Each station is restricted to completing the tasks in a time interval known as the cycle time. The time spent by each station in completing the required tasks is known as the process time. In the simple assembly line balancing problem, the objective is to minimize the total idle time within the station. This is equivalent to minimizing the number of stations for a given cycle time or minimizing the required cycle time when the number of stations is known. A thorough description of this problem is given by Kilbridge and Wester (1961).

Since the introduction of the simple line balancing problem, however, many variations have appeared in the literature. Side constraints have been added to the problem to force certain tasks to appear at a given station (Gunther, Johnson, and Peterson (1983)), to restrict placing two tasks in the same station (Mitchell (1957)), and to place restrictions on uneven idle times at the stations. Dar-El (1978) allows for mixed models of the same product to be produced on the same line. Others allow tasks with large process times to be divided onto two parallel stations to improve the total line performance (Pinto, Dannenbring, and Khumawala (1975)). These latter improvements can be incorporated

into the formulation of the simple assembly line model if the modifications can be anticipated. Further improvements have been suggested that minimize costs of setting up stations and operating the assembly line. These costs have been incorporated to consider alternative processing methods (Pinto, Dannenbring, and Khumawala (1983)) and assembly line layouts for robotic applications (Graves and Lamar (1983)). These cost-based models have been termed assembly line design problems by Baybars (1986). These additional objective criteria for assembly line balancing problems also suggest the application of multiple criteria methods including goal programming (Gunther, Johnson, and Peterson (1983)).

Clearly the assembly line balance problem has received much attention in the literature in recent years and many algorithms exist to find the balance for an assembly line. Many of the heuristics are reviewed by Mastor (1970) and Talbot, Patterson, and Gehrlein (1986), while Baybars (1986) reviews the exact algorithms for the simple assembly line balancing problem. Each of these approaches are directed at solving a particular formulation of the simple assembly line balancing problem. We discuss an optimization methodology for solving many variations of the assembly line balancing problem. The approach is an implicit enumeration technique along the lines of Talbot and Patterson (1984). We discuss how to adapt the algorithm, based on Balas' (1965) procedure, to several variations.

In the next section we review the assembly line balancing model and several variations, some of which involve modification of the constraint set and / or the objective function. The third

section contains the general solution methodology, followed in Section 4 by explanations of methodological differences due to different formulations. In Section 5, we describe computational experience. We offer a summary and conclusions in Section 6.

2. The Assembly Line Balancing Model

Here, we treat the assembly line balancing problem in its network representation. For example, consider a problem from Jackson (1956) in Figure 1. The nodes represent the tasks to be assigned to stations and the arcs represent precedence relations. Each node has its associated task's processing time. The specified cycle time determines the work capacity of each station. We now present the formulation for the simple assembly line balancing problem. Following it, we introduce variations of the model. Our notation is to let:

- n be the number of tasks,
- K be the maximum number of stations,
- X_{ik} be a zero-one variable representing whether task i is in station k (=1) or not (=0),
- S_k be a zero-one variable representing whether station k is used (=1) or not (=0),
- t_i be the nonnegative process time for task i,
- C be the nonnegative cycle time, i.e., bound on the total task time in each station,
- B_k be the nonnegative bound on the number of tasks in station k,
- T be the total process time for all tasks, $(T = \sum_{i=1}^{n} t_i)$.

Without loss of generality, we assume that the ordering of the tasks is such that lower numbered tasks always precede higher numbered ones. We may now state the mixed integer program for the

simple assembly line balancing problem (ALB) as:

$$(1) \qquad \text{MIN} \quad \sum_{k=1}^{K} S_k$$

$$(2) \qquad (1/n) \left(\sum_{i=1}^{n} X_{ik} \right) \leq S_k \quad \text{, for each station } k = 1, \ldots, K,$$

$$(3) \qquad \sum_{k=1}^{K} X_{ik} = 1 \quad \text{, for each task } i = 1, \ldots, n,$$

$$(4) \qquad \sum_{i=1}^{n} t_i X_{ik} \leq C \quad \text{, for each station } k = 1, \ldots, K,$$

$$(5) \qquad \sum_{k=1}^{K} k X_{ik} \leq \sum_{k=1}^{K} k X_{jk} \quad \text{, for each i,j preference pair, where i is the immediate predecessor to j,}$$

$$(6) \qquad X_{ik} = 0,1 \quad \text{, } i=1,\ldots,n;\ k=1,\ldots,K,$$
$$S_k = 0,1 \quad \text{, } k=1,\ldots,K.$$

The objective function (1) is to minimize the number of stations. Constraint set (2) forces S_k to 1 if any task is assigned to station k. Constraint set (3) ensures that each task is assigned to a station. That the total task time in each station does not exceed the cycle time is enforced by (4).

Constraint set (5) provides the precedence relationships. In (6), the integrality restrictions on the decision variables are explicitly stated.

2.1 Variations on the Constraint Set

To consider tasks that must be assigned to a particular station we add the constraint:

(7) $X_{ik} = 1$, for the specific i, k requirement.

This also allows the removal of the i^{th} constraint from set (3). In fact, X_{ik} can be substituted out of ALB. To consider the case where two tasks must appear together, we add the constraint set:

(8) $X_{ik} - X_{jk} = 0$, for the i,j pair, for each station $k = 1, \ldots, K.$

Another variation we consider is the case where two tasks cannot appear in the same station. For this, we add the constraint set:

(9) $X_{ik} + X_{jk} \leq 1$, for the i,j pair, for each station $k = 1, \ldots, K.$

The extension of (9) to several tasks is clear. A final variation in this set is to limit the number of tasks that may be assigned to a station. This condition may be specified by:

(10) $\sum_{i=1}^{n} X_{ik} \leq B_k$, for $k = 1, \ldots, K.$

If there is a minimum number of tasks to be assigned to a station, then the summation in (10) may be bounded from below by a G_k. Other minor variations may be added in similar fashion.

2.2 Minimize the Cycle Time

A common variation on the basic model is to fix the number of stations and minimize the cycle time. Procedurally, this can be accomplished by solving the basic formulation at successively higher cycle times until the desired number of stations is obtained. The formulation may also be revised by allowing the cycle time, C, to be a nonnegative decision variable, replacing the objective function (1) with:

(11) MIN C,

eliminating constraint set (2) and specifying $C \geq 0$. Even though this problem now appears smaller than the basic formulation, the removed variables and constraints are implicitly considered by many of the solution techniques in the literature (Baybars (1986)).

2.3 Balance the Work Load

Under certain contractual and supervisory situations, it is desirable to even the workload at the individual stations when the number of stations and the cycle time is known. Thus, the total time of the work content, or equivalently, the idle times of all the individual work stations should be as "close" as possible. A method for handling this variation is to include deviational variables (D_k^+, D_k^-) that represent the difference between the ideal process time of each station, T/K, and the actual processing time of each stations. The objective becomes:

$$(12) \qquad \text{MIN} \quad \sum_{k=1}^{K} \left(D_k^+ + D_k^- \right).$$

Constraint (2) is removed and the following constraints must be added:

$$(13) \qquad \sum_{i=1}^{n} t_i X_{ik} + D_k^- - D_k^+ = T/K , \quad \text{for each } k = 1, \ldots, K,$$

$$(14) \qquad D_k^- , D_k^+ \geq 0 , \quad \text{for each } k = 1, \ldots, K.$$

It is possible to weight the original objective (1) with that expressed in (12). Alternatively, it is possible to solve the original problem to determine the optimal number of stations, then to use this formulation to disperse the idle time among the stations. When the cycle time is flexible, (13) replaces (4).

2.4 Station Costs

It has become a concern of how to utilize work stations in a cost effective manner when there exist alternate facilities (Graves and Lamar (1983)). It is also possible that an older assembly line may have to be retooled or redesigned to become economically feasible to retain. This requires that the cost of the upgraded system be as low as possible when fitting a new product into an older assembly line. Two costs become evident, the fixed cost of using a station, and the variable cost of performing a task at a specific station. For this variation we introduce two new parameters. Let f_k be the fixed cost of opening or constructing station k. Let v_{ik} be the cost of assigning task

i to station k. The new objective function becomes:

$$(15) \qquad \text{MIN} \quad \sum_{k=1}^{K} f_k S_k + \sum_{i=1}^{n} \sum_{k=1}^{K} v_{ik} X_{ik}$$

No changes to the constraints of the basic problem are required. This formulation selects the stations to use and assigns tasks to the stations.

2.5 Interaction Costs

Another cost variation is inspired by the work done in Computer Assisted Process Organization (CAPO) (Klein, Beck and Konsynski (1985)). The constraint set of the CAPO problem is of a structure similar to that of the assembly line balancing problem. The objective is different than common formulations however, because costs are associated with placing items into the same group with one another. Thus the objective is similar to those in certain cluster analysis problems (Klein and Aronson (1988), Aronson and Klein (1988)), where the objective is to place observations into classifications (tasks into stations) such that the sum of pairwise dissimilarities (costs or distances) of observations within clusters (interactions or interference among tasks) is minimized.

This formulation is useful in assembly line balancing models to encourage similar tasks (i.e. grinding and sanding) to be assigned to the same station, and to discourage dissimilar, or conflicting tasks (i.e. baking and painting), from being assigned to the same station. For this particular modification, it is the objective function rather than the constraint set that attempts to

enforce such side conditions. There are two specific line balancing situations that motivate this cost modified variation of the problem. The first is to split the tasks of a line into segments that should be grouped together, but not necessarily in the same station, because the final product is segmented into different components requiring different types of operations (i.e. for metal, wood, leather, final assembly, etc.). See Gaither (1986) for an example. A second situation is when a series oriented or parallel oriented line is desired by design. When the tasks are to be assigned to stations in series by columns according to the network representation, then, these steps should have low interaction costs specified. The interaction costs of tasks from start to finish through the network should be of increasing magnitude. When a parallel oriented line is desired, the interaction costs are high for pairs of tasks from top to bottom in the network, and low for pairs of tasks from left to right.

To accomplish this objective requires the addition of the 0-1 variable Y_{ij} to indicate when two tasks i and j are assigned to the same station ($Y_{ij} = 1$) or not ($Y_{ij} = 0$). Also, let d_{ij} be the <u>positive</u> dissimilarity measure of tasks i and j, that is d_{ij} represents the relative measure of nondesirability of having task i in the same station with task j. Including the fixed cost of using a station from (15) above, the objective function (1) becomes:

$$(16) \quad \text{MIN} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} Y_{ij} + \sum_{k=1}^{K} f_k S_k ,$$

The following constraint sets must be added to relate the station membership to the objective function:

$$(17) \quad Y_{ij} \geq X_{ik} + X_{jk} - 1 \text{ , for each i,j pair in each group,}$$
$$\text{i.e., for } i=1, \ldots, n-1;$$
$$j = i+1, \ldots, n; \; k = 1, \ldots, K.$$

$$(18) \quad 0 \leq Y_{ij} \leq 1 \text{ , for } i=1, \ldots, n-1;$$
$$j = i+1, \ldots, n.$$

Note, the Y_{ij} need not be restricted to be a 0-1 variable when all the d_{ij} are positive. The following constraints must be added if any of the d_{ij} are nonpositive:

$$(19) \quad Z_{ij} \geq (X_{ik} + X_{jk})/2 \quad \text{for } i = 1, \ldots, n-1;$$
$$j = i+1, \ldots, n; \; k = 1, \ldots, K.$$

$$(20) \quad Y_{ij} \leq Z_{ij} \quad \text{for } i = 1, \ldots, n-1;$$
$$j = i+1, \ldots, n,$$

$$(21) \quad Y_{ij} = 0, 1 \quad \text{for } i = 1, \ldots, n-1;$$
$$j = i+1, \ldots, n.$$

In addition, the objective (16) must be changed to:

$$(22) \quad \text{MIN} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (d_{ij} Y_{ij} + M Z_{ij}) + \sum_{k=1}^{K} f_k S_k ,$$

where M is a large number.

2.6 Multiple Criteria

Any combination of the cost measures can be made without much conceptual difficulty because they represent the same units of measure. If, however, multiple objectives of different measures exist, then a goal program may be used (Gunther, Johnson, and Peterson (1983)). The goal program may be a weighted goal program or a priority goal program (Ignizio (1982)).

3. General Solution Method

We now discuss an algorithm for solving the assembly line balancing problem (ALB), including its variations. Our algorithm is an extension of the one discussed by Klein and Aronson (1988). It is also an extension of Balas' (1965) implicit enumeration algorithm that handles multiple branches from each solution node in the branch and bound tree. Though not required, a depth-first search strategy is used. The level, or depth, of a node in the tree represents the task under consideration; the branches represents station assignment. The complete enumeration of the tree is shown in Figure 2 for the example problem in Figure 1. However, our algorithm prunes many branches by determining the feasibility of a solution from precedence relations, cycle time limits, tight bounds, and bounds on which branches are to be explored. For the variations, additional bounds may be derived, and, the feasibility of additional solutions may automatically be tested.

3.1 Additional Notation

We need the following additional notation to state the algorithm:

- p - a pointer to the current depth in the tree, corresponding to the current task,
- m - station membership indicator (from 1 to K),
- n(m) - the number of tasks in station m,
- t(m) - the amount of time required by the tasks in station m,
- L_i - the first feasible station for task i, imputed by the precedence relationships and the cycle time,

R_i - the last feasible station for task i, imputed by the precedence relationships and the cycle time,

Z - the value of the objective function of the incumbent,

X_{IK} - the vector containing the values of all X_{ik} at the current incumbent.

3.2 The Steps of the Basic Solution Method

We first present a general statement of the algorithm and then discuss implementation issues for the basic formulation ALB.

STEP 0: Initialize all vectors. These include L, R, t(m), and the data structures used for storing bounds and constraints. Initialize the pointer p to a depth of 0. Initialize the incumbent to infinity or to a solution found by a line balancing heuristic.

STEP 1: Increment the search depth by placing the next task (p = p + 1) into its first feasible station (m = L_p). Update the time used in the station chosen (t(m) = t(m) + t_p).

STEP 2: Test for feasibility. If any constraint is violated, or if the tasks that have the current task as a predecessor cannot be assigned in the time remaining in the following stations, the current assignment is infeasible.
Go to Step 5 on any infeasibility.

STEP 3: Test for suboptimality. If the current objective value plus any bound is less than the incumbent objective value (Z) then proceed to Step 4. Otherwise go to Step 5.

STEP 4: If the entire set of tasks has been assigned (p=n) then update the incumbent and appropriate bounds, else return to Step 1. If the objective is equal to the theoretical

minimum number of stations (= <T/C>, where <.> means to round up to the next highest integer), then STOP, the current incumbent is optimal.

STEP 5: Attempt a depth fathom. If the current task is not assigned to the last feasible group ($m < R_p$) then proceed to Step 6. Otherwise update t(m), $t(m) = t(m) - t_p$, retract one level in the depth of the search, $p = p-1$, and STOP if $p < 1$, which indicates that the current incumbent is optimal. If no incumbent has been found, then the subproblem is infeasible. Repeat Step 5 to continue fathoming.

STEP 6: Place the current task in the next station. Update $t(m) = t(m) - t_p$. Reset the station indicator $m = m + 1$. Update $t(m) = t(m) + t_p$. Return to Step 2.

A new incumbent is found only at depth n. The cycle time constraints (4) may be quickly tested to determine if a problem is infeasible. The precedence constraints (5) in conjunction with (4) determines the earliest (L_i) and latest (R_i) station in which a task may appear.

3.3 Implementation Issues

The general algorithm will now be explored more fully in terms of the basic formulation and the example problem shown in Figure 1. The issues of concern mostly involve bounding, feasibility, heuristic initial solutions, and data structures.

3.3.1 Bounding

The bounding routines used for the basic problem involve the use of upper and lower bounds on the station assignment based on the precedence relations (5) and the cycle time capacity constraints (4). These are introduced by Patterson and Albracht (1975). Intuitively, if a task is embedded in a precedence structure, then there are a certain number of tasks that must be completed prior to the beginning of the task and a certain set of tasks that must be accomplished after the task. In the example problem shown in Figure 1, task 9 must be preceded by tasks 1, 3, 4, and 5. These required pretasks require a total of 19 time units and task 9 requires 3 units. Thus, if the desired cycle time of the line were 10 time units, task 9 cannot possibly be assigned to station 1 or 2. A similar argument holds for the upper bound on the station assignment by considering the time of the tasks that must follow a particular task.

We define a required predecessor of i, to be a task that must be assigned before being able to assign task i. A similar definition applies to a required follower of i. Let P_i be the set of all required predecessors of i, and F_i be the set of all required followers of i. The upper (R_i) and lower (L_i) bounds on the station to which task i may be assigned are computed in Step 0 as described below. Based on the cycle time constraints, we set R_i to be the last possible station into which task i may be assigned as:

$$(23)\quad R_i = \text{Max}\ \{\ r = K,..,1 \mid t_i + \sum_{j \epsilon F_i} t_j - (K-r+1)\ C \leq 0\ \},$$

If no such value exists in (23), we assign the value K.

We define L_i similarly by:

$$(24)\quad L_i = \text{Min}\ \{\ s = 1,..,K \mid t_i + \sum_{j \epsilon P_i} t_j - s\,C \leq 0\ \},$$

If no such value exists in (24), we assign the value 1. Note that the definition of L_i and R_i imply that constraint set (3) may be rewritten as:

$$(25)\quad \sum_{k=L_i}^{R_i} X_{ik} = 1, \qquad \text{for } i = 1, \ldots, n.$$

These bounds may then be updated as new incumbent solutions are found. Since an incumbent solution provides an upper bound on the maximum number of stations required, then, one less than the number of stations of the incumbent is the maximum number of stations allowed in any improving solution. This station assignment bound update (for R_i) may be accomplished in Step 4 by subtracting the difference between the old incumbent and the new incumbent objective value from each R_i and by revising K accordingly. If while optimizing, $R_i < L_i$ for any task i, then the algorithm may stop with an optimal incumbent because there no longer exists a solution having less stations than the incumbent.

3.3.2 Feasibility

Feasibility tests are made during Step 2. Cycle time constraint tests are made by comparing the value of the total task time in the current station (t(m)) to the cycle time value. Precedence constraints are checked by ensuring that the assignment of task p to station m does not violate any of the stored precedences. Any violation leads to a fathom. For example, if

task 3 in Figure 1 were assigned to station 2 then it would be infeasible to assign task 9 to station 1. The corresponding node on the enumeration tree and all succeeding nodes are eliminated from consideration. Many of these are implicitly fathomed through the L_i and R_i in directing the search strategy.

Two other feasibility tests may lead to node fathoms. The first being a test of the upper and lower bounds on each task assignment. The situation of when $R_i < L_i$ was discussed earlier. The second feasibility test involves using the times computed for the original upper and lower bound determination. Compare the time remaining in the station assignment to the final station (m to K) at the current task depth (p). If the time required by the followers of task p cannot fit in the remaining time, the current node on the enumeration tree and all nodes below it are infeasible. Using the example problem, if the cycle time is 10, task 8 is assigned to station K, and tasks 7 and 9 are assigned to station K-1, then there is not sufficient time remaining in stations K-1 and K to assign tasks 10 and 11. These times of successive tasks are computed in Step 0 during the determination of the L and R vectors. Let these values be stored in a vector of length n call FOLLOW. Let the total times of the required preceding tasks be stored in a vector called PRIOR. Using these vectors, logical checking occurs for feasibility only within the algorithm's repetitive loop, thus requiring little computational overhead.

3.3.3 Heuristic Starting Solutions: Fast Starts

Rather than starting the implicit enumeration algorithm with no incumbent, i.e., having an infinite objective value, an efficient and effective heuristic to find a good initial solution should speed up the convergence of the algorithm. For the simple assembly line balancing problem, we implemented the minimum task time selection rule heuristic. See Mastor (1970) for a discussion of the relative merits of the various selection rules. In our implementation, to modify the selection rule requires changing only a few lines of code. In fact, we could select the selection rule by an input flag.

For problem variations, the heuristic takes additional side constraints into consideration. For the variations having interaction costs, the heuristic attempts to minimize the number of stations used on its first pass. An improving routine shuffles tasks pairwise among the stations, including the unused ones, in an attempt to minimize interaction costs. It considers the fixed-cost of opening a new station when such costs are present.

Candidate lists constructed via the precedence relationships were implemented in the heuristic and its improvement routine. In the computational tests described later, the execution CPU times of these routines were negligible.

3.3.4 Data Structures

It is always more efficient to access vectors rather than arrays when computer time is a major consideration. Thus all constraints and parameters should be reduced to vectors whenever possible. For the basic problem this includes the variable

vector, the bounds on station assignments represented by R and L, the precedence constraints, the and bound vectors PRIOR and FOLLOW.

The variables X_{ik} may be represented by a single vector of length n. The ith position in the vector represents the integer value of the station assigned to the ith task. Thus, X_{ik} is reduced to X_i where the value corresponds to the assigned station number. The bound vectors (R and L) and the vectors containing the total succeeding task times and preceding task times are computed prior to Step 1 and used during the iterations of Steps 1 through 6.

Costs of a current node in the solution algorithm are stored in a vector of length n. The position in the vector indicates the cost of the enumeration at the indicated depth. Thus, as a higher numbered task is added to the enumeration at a lower level in the tree, the cost is determined as the cost found at the previous depth plus the incremental cost acquired by assigning only the current task. In this fashion, the computational effort of cost determination is minimized.

The precedence relationships are preserved by three vectors that store the backward star representation of the arcs. The first is a vector of length n. It contains the number of immediate predecessors for each task, denoted PNBR. Thus in Figure 1 the vector is (0,1,1,1,1,1,1,1,3,1,2). The second vector, denoted PREC, is of length at least equal to the number of precedence constraints. It contains a simple list of the immediate predecessors of each task for as many predecessors as there exist. For the problem shown in Figure 1, PREC is

{1,1,1,1,2,6,7,3,4,5,9,8,10}. The third vector, denoted POS, is of length n and contains the position of the first predecessor for each task. For the example problem, POS is {1,1,2,3,4,5,6,7,8,11,12}. These vectors are initialized during Step 0 when reading in a problem. During each iteration, the feasibility test in Step 2 need only consider the entries from POS through POS + PNBR - 1 of the single vector PREC.

3.4 Example

For the example problem shown in Figure 1, assume a desired cycle time of 21. Also assume that the fast start procedure has found a solution of 4 stations so that the algorithm will search for a solution of 3 stations or less. Figure 3 represents the enumeration tree developed by the algorithm in solving this problem.

In Step 0, the bound vectors are determined to be R = {1,3,3,3,3,3,3,3,3,3,3} and L = {1,1,1,1,1,1,1,1,2,2,3}. The precedence vectors are determined as above. Set t(k) = 0 for all k. The incumbent is set to the solution found by the heuristic; p is set to zero. This represents starting node 0. Though they are not used in this example, the vectors PRIOR and FOLLOW are set to {0,6,6,6,6,8,10,16,19,22,42} and {40,17,12,12,12,15,9,4,9,4,0} respectively. In Step 1, p is incremented to 1 and the first task is placed in station number m=1, at node 1 of the tree. Step 2 finds no infeasibility and Step 3 does not find the current solution to be worse than the incumbent. The assignment of tasks to stations is not complete so we go back to Step 1.

This process repeats, adding successive tasks to station one by adding nodes 2, 3, 4, and 5 to the enumeration tree in Figure 3. Task 6 is assigned to station 1 in node 6 of the tree. However, the cycle time is exceeded in station 1, and node 6 is fathomed, eliminating all possible lower nodes. Step 6 then assigns task 6 to station 2 at node 7 and returns control to Step 2. The algorithm fathoms on cycle time infeasibility at nodes 8 and 10. Nodes 12 and 14 in the tree are implicitly enumerated and fathomed (denoted as smaller nodes) because $L_9 = L_{10} = 2$. At node 15, only task 11 remains. It is assigned to the third station in Step 1 (recall, $L_{11} = 3$). Thus, nodes 16 and 17 were implicitly enumerated and fathomed. Finally at node 18 a feasible solution using three stations is found. The incumbent is updated in Step 4. The algorithm stops because the theoretical minimum number of stations is reached. The optimum found assigns tasks 1 - 5 to station 1, tasks 6 - 10 to station 2, and task 11 to station 3.

4. Alterations in the Solution Method for Variations

Certain changes to the algorithm may result from the variations presented in Section 2. When required, the changes are minor, but the computational efficiency of the algorithm are sometimes unpredictable.

The variations on the constraint set present no difficulties. They all fit into the basic algorithm in Step 2. Vectors such as those used for the precedence relations should also be used for constraints of type (7 - 9). Techniques similar that presented for the cycle time limits may be developed to update the L_i and R_i using the station size restrictions (10) as follows:

$$(26)\quad R_i = \text{Max}\ \{\ r = K,\ldots,1 \mid \quad 1 + |F_i| - \sum_{k=r}^{K} B_k \le 0\ \},$$

$$(27)\quad L_i = \text{Min}\ \{\ s = 1,\ldots,K \mid \quad 1 + |P_i| - \sum_{k=1}^{s} B_k \le 0\ \},$$

Where $|F_i|$ represents the number of elements in set F_i. If no such value exists in (26), we assign the value K, if no such value exists in (27), we assign the value 1.

If the station size restriction is present, we use the minimum of the right hand side of the expressions in (23) and (26) for determining R_i, and the maximum of the right hand sides of the expressions in (24) and (27) for determining L_i. If a constraint (7) conflicts with the upper and lower bounds on station assignment, the algorithm can stop.

Minimizing the cycle time (11) for a specific number of stations can be solved by the method, but the use of the strong

upper and lower bounds is weakened because cycle time is not determined prior to the use of the algorithm. An initial solution found by a heuristic, or the first feasible solution found, should be used to determine the R and L vectors. When a new incumbent is found, it may be worthwhile to recompute the R and L vectors. At any node in the enumeration tree, tighter objective function bounds may be found by placing the unassigned task with the highest task time into the station with the lowest total task time assigned. If this result exceeds the objective value of the incumbent, Step 3 may conduct a fathom. FOLLOW and PRIOR also provide bounds, though different from those of the basic formulation. In this case, the time of the tasks already assigned to the current station (m) plus the time of the tasks already assigned to all the previous stations plus the time of all tasks that must precede the current task (p) divided by m gives a lower bound on the cycle time at the current node in the enumeration. A similar argument holds for the time following p and m. If either of these bounds exceeds the objective value of the incumbent solution, the node may be fathomed.

Balancing the work load (12 - 14) does allow for the use of the initial upper and lower bounds but will not allow tightening of these bounds on station assignment since the number of stations remains fixed throughout the procedure.

The fixed costs of stations in the objective (15) provide unique opportunities. The initial upper and lower bounds on station assignments apply, but are not updated in Step 4 since the number of stations is fixed. Feasibility tests are the same as

those in the basic algorithm. Additional bounds are also available.

The cost at each node only includes the costs of the tasks already assigned. Bounds exist on the unassigned tasks. Lower bounds on the cost of assigning the remaining tasks (>p) to work stations may be computed using the fixed and variable portions of the station costs. To compute the variable portion we sum the lowest cost feasible assignment for each task while taking into consideration time limits, capacity limits, any lower limits on the station, as well as any precedence relationships.

The fixed-cost portion may be determined as follows. The available time remaining in the existing stations is

$$T_R = \sum_{k=1}^{K} \left(S_{ik}\, C - \sum_{i=1}^{n} t_i\, X_{ik} \right) \tag{28}$$

Then, the amount of time exceeding the available time remaining required by tasks p+1, ..., n is

$$T_N = \sum_{i=p+1}^{n} t_i - T_R \tag{29}$$

If T_N is positive, then, at a minimum, $\langle T_N/C \rangle$ additional stations are required. We then add to the bound the lowest $\langle T_N/C \rangle$ fixed costs of the unused stations. The p^{th} value of FOLLOW may be updated accordingly.

The use of the pairwise interaction costs (16 - 18) in determining bounds is similar to the that of the station costs just discussed. The only difference is in the computation of the

variable bounds. In this case, costs will be incurred by interactions among the unassigned tasks as well as any interactions the unassigned tasks achieve when assigned to a station. These bounds can be very tight and are discussed in depth by Aronson and Klein (1988). They compute the minimum interactions among the unassigned nodes by first computing the minimum number of interactions at each level p. The bounds for each level p are found in reverse order by adding the lowest interaction costs in the cost matrix d_{ij} for the minimum number of interactions to the costs found for p+1. This is a constant vector and may be computed in Step 0 to avoid computations during each iteration. The portion of the bound due to interactions of unassigned tasks to assigned tasks is computed by assigning each remaining task (i>p) to each feasible station and selecting the lowest of the interactions found. These bounds are added to the cost found in Step 3 of the algorithm determine if a suboptimality fathom is possible.

5. Computational Experience

All of the line balancing routines were incorporated into the implementation GROUPS2. The code, written in FORTRAN 77, was compiled and tested on the Southern Methodist University IBM 3081-D24 running the VM/CMS Operating System. The FORTVS2 compiler set at optimization level 3 was used. All reported execution times are in CPU seconds. They include the time to run the algorithm plus the negligible time to find an initial solution using the described heuristics.

The convergence properties of the algorithm are of great concern because the algorithm could evaluate the maximum number of solution nodes in the tree, $\sum_{i=0}^{n} K^i$. Convergence is improved through the obtainment of tight bounds and the use of the heuristic start procedure.

Various times have been cited for balance line algorithms in the literature. Many of these are reported in Baybars' (1986) review. For our study, we selected the classic problems of Jackson (1956), Mitchell (1957), and Tonge (1961) for testing of the basic algorithm on solving the simple assembly line balancing problem. Note that Baybars (1986) suggests that Tonge's (1961) 70 task problem has become a benchmark test problem over the last 25 years.

In Table 1, we show the solution CPU time required to solve the simple assembly line balancing problems cited. The solution CPU time required to obtain an optimum for six different cycle times for all three problems never exceeded .03 CPU seconds. Baybars (1986) summarizes the computational results of Dar-El

(1978) and Wee and Magazine (1981) on solving these problems for various cycle times on an IBM 370/158. Even allowing for computer differences, our implementation dramatically outperforms them both on the largest problems. Including the extra constraint variations into the implementation of the algorithm added no perceptible computation time.

In terms of convergence with the first set of tests, for the first model, a maximum of 1941 nodes were evaluated, out of a possible $1.11 * 10^{11}$nodes; for the second and third models, the worst case peeformances were 1694 out of $1.26 * 10^{24}$ nodes, and 172 out of $3.81 * 10^{75}$ nodes. The method evaluates only a very small percentage of the maximum total number of solution nodes. Many solution nodes are fathomed based on feasibility.

In Table 2, we report the solution CPU times required for solving cost variation models using randomly generated interaction costs and fixed costs of station use for the model of Mitchell (1957). Though the times are much worse than for the simple case (.23 - 19.38 CPU seconds versus less than .005 - .03 CPU seconds), it is important to note that these are extremely difficult problems. It is possible, with our methodology, to solve such problems to optimality within a reasonable amount of computational time. In general, the cost of obtaining a solution is insignificant to the cost of developing the manufacturing system, but, if desired, bounds combined with solution objective value tolerances may be used to determine an early termination.

For this second set of tests, the most nodes evaluated was about 300,000, but that was only $2.4 * 10^{-13}$ % of the total number

of nodes for that problem. In all our tests, the percentage of solution nodes evaluated by GROUPS2 ranged from 0 % to $1.7*10^{-6}$ %.

6. Summary and Conclusions

We have presented a general mathematical programming formulation for the assembly line balancing problem. Extensions, including one having pairwise interaction costs in the objective function, were also discussed.

We also presented an efficient, implicit enumeration algorithm for optimizing the simple problem and its extensions. We have described specialized bounding techniques and other implementation issues. The development and implementation of specialized bounding rules derived from the precedence, cycle time and other capacity constraints dramatically decrease the number of subproblem nodes that need to be evaluated by the implicit enumeration algorithm. The computational results show that our implementation of the enumeration method is efficient and viable for solving the simple problems reported in the literature and their complicated extensions. The method compared favorably to other algorithms.

Future research in this area should be directed toward improvements in the methodology, especially toward improving implementations of methods for solving the presented extensions and others. Other work should encompass detailed development of multiple criteria models and their solutions, parallel implementations, and the development of further extensions of the model to encompass other real-world situations.

REFERENCES

Aronson, J.E. and G. Klein. 1988. "The Solution of Computer Assisted Process Organization Problems with a Cluster Analysis Algorithm," College of Business Administration Working Paper Number 87-241, The University of Georgia, Athens, GA, 30602, February (revised).

Balas, E. 1965. "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, 13, 3, 517-546.

Baybars, I. 1986. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem," Management Science, 32, 8, 909-932.

Dar-El, E.M. 1978. "Mixed-Model Assembly Line Sequencing Problems," OMEGA, 6, 4, 313-323.

Gaither, N. 1986. Production and Operations Management, 3rd edition, Dryden Press, NY.

Graves, S. and B. Lamar. 1983. "An Integer Programming Procedure for Assembly System Design Problems," Operations Research, 31, 3, 522-545.

Gunther, R.E., G.D. Johnson, and R.S. Peterson. 1983. "Currently Practiced Formulations for the Assembly Line Balance Problem," Journal of Operations Management, 3, 209-221.

Ignizio, J.P. 1982. Linear Programming in Single and Multiple-Objective Systems, Prentice-Hall, Englewood Cliffs, NJ.

Jackson, J.R. 1956. "A Computing Procedure for a Line Balancing Problem," Management Science, 2, 3, 261-271.

Kilbridge, M.D. and L. Wester. 1961. "The Balance Delay Problem," Management Science, 8, 69-84.

Klein, G. and J.E. Aronson. 1988. "Optimal Clustering: A Model and Method," E. L. Cox School of Business Working Paper Number 86-121, Southern Methodist University, Dallas, TX, 75275, February (revised).

Klein, G., P. Beck, and B. Konsynski. 1985. "Computer Aided Process Structuring Via Mixed Integer Programming," to appear in Decision Sciences; Edwin L. Cox School of Business Working Paper Number 85-111, Southern Methodist University, Dallas, TX, 75275.

Mastor, A. 1970. "An Experimental Investigation and Comparative Evaluation of Production Line Balancing Techniques," Management Science, 16, 22, 728-745, July.

Mitchell, J. 1957. "Computational Procedure for Balancing Zoned Assembly Lines," Research Report 6-94801-1-R3, Westinghouse Research Laboratories, Pittsburgh, PA.

Patterson, J.H. and J.J. Albracht. 1975. "Assembly Line Balancing: O-1 Programming with Fibonacci Search," Operations Research, 23, 1, 166-174.

Pinto, P.A., D.G. Dannenbring and B.M. Khumawala. 1975. "A Branch and Bound Algorithm for Assembly Line Balancing with Paralleling," International Journal of Production Research, 13, 2, 183-196.

Pinto, P.A., D.G. Dannenbring and B.M. Khumawala. 1983. "Assembly Line Balancing with Processing Alternatives: An Application," Management Science, 29, 7, 817-830.

Talbot, F.B. and J.H. Patterson. 1984. "An Integer Programming Algorithm with Network Cuts for Solving the Assembly Line Balancing Problem," Management Science, 30, 1, 85-99, January.

Talbot, F.B., J.H. Patterson, and W.V. Gehrlein. 1986. "A Comparative Evaluation of Heuristic Line Balancing Techniques," Management Science, 32, 4, 430-454, April.

Tonge, F.M. 1961. A Heuristic Program of Assembly Line Balancing, Prentice-Hall, Englewood Cliffs, NJ.

Wee, T.S. and M.J. Magazine. 1981. "An Efficient Branch and Bound Algorithm for Assembly Line Balancing - Part 2: Maximize the Production Rate," Working Paper #151, University of Waterloo, Waterloo, ONT., Canada.

Problem	Number of Tasks	Cycle Time	Optimal Number of Stations	Solution CPU Time
Jackson(1956)	11	7	8	.03
		9	6	.01
		10	5	<.005
		13	4	<.005
		14	4	<.005
		21	3	.01
Mitchell(1957)	21	14	8	.02
		15	8	<.005
		21	5	.01
		26	5	<.005
		35	3	.01
		39	3	<.005
Tonge(1970)	70	346	11	<.005
		358	11	<.005
		364	11	.01
		410	9	<.005
		468	8	<.005
		527	7	.01

Table 1: Computational Results of the Branch and Bound Algorithm Solving Simple Assembly Line Balancing Problems. All tests were performed on the Southern Methodist University IBM 3081-D24 running the VM/CMS Operating System, and the FORTVS2 compiler at optimization level 3. The Number of Tasks is the problem size. The Cycle Time is a problem parameter. The Optimal Number of Stations is found by the algorithm. The Solution CPU Times are in CPU seconds. A CPU time of <.005 indicates that the solution CPU time was less than .005 seconds.

Cycle Time	Number of Stations Allowed	Optimal Number of Stations	Minimum Number of Stations	Solution CPU Time
35	4	4	3	2.93
35	4	4	3	2.47
35	5	4	3	19.38
23	5	5	5	0.23
21	6	6	5	3.40
21	6	6	5	5.79
15	8	8	8	0.66
14	9	9	8	15.34

Table 2: Computational Results of the Branch and Bound Algorithm Solving Assembly Line Balancing Problems with Random Interaction Costs and Fixed Costs of Station Use. The problem solved is the 21 task Mitchell (1957) model. All tests were performed on the Southern Methodist University IBM 3081-D24 running the VM/CMS Operating System, and the FORTVS2 compiler at optimization level 3. The Number of Stations Allowed is a parameter. The Optimal Number of Stations is found by the algorithm. The Minimum Number of Stations is the theoretical bound on the number of stations. The Solution CPU Times are in CPU seconds.

Figure Descriptions

Figure 1: Assembly Line Balancing Example Problem due to Jackson (1956). The nodes represent the tasks to be assigned to stations and the arcs represent precedence relations. Each node has its associated task's processing time. The specified cycle time determines the work capacity of each station. There are 11 tasks. The total task time is 46 units.

Figure 2: Complete Enumeration Tree for the Assembly Line Balancing Example Problem Shown in Figure 1 for a Cycle Time of 21 and a Maximum of 5 Stations. Each level of depth corresponds to a task, 1, ..., 11. The leftmost node (subproblem) corresponds to assigning the task at that level to the first station, the next node to the right corresponds to assignment to the second station, and so on to the fifth station. The minimum number of stations is 3. There are 61,035,156 nodes in this tree.

Figure 3: The Enumeration Tree for the Assembly Line Balancing Example Problem Shown in Figure 1 for a Cycle Time of 21 and a Maximum of 5 Stations. The problem was solved by the implicit enumeration algorithm. Inside each node is the node number. Outside each node in parenthesis are the (task number, station assignment). Large nodes are explicitly enumerated. Small nodes are implicitly enumerated. A heuristic has found an initial solution using 4 stations at node 0. The optimum is found at node 18 (indicated by a *) with the solution of tasks 1 - 5 in station 1, tasks 6 - 10 in station 2, and task 11 in station 3.

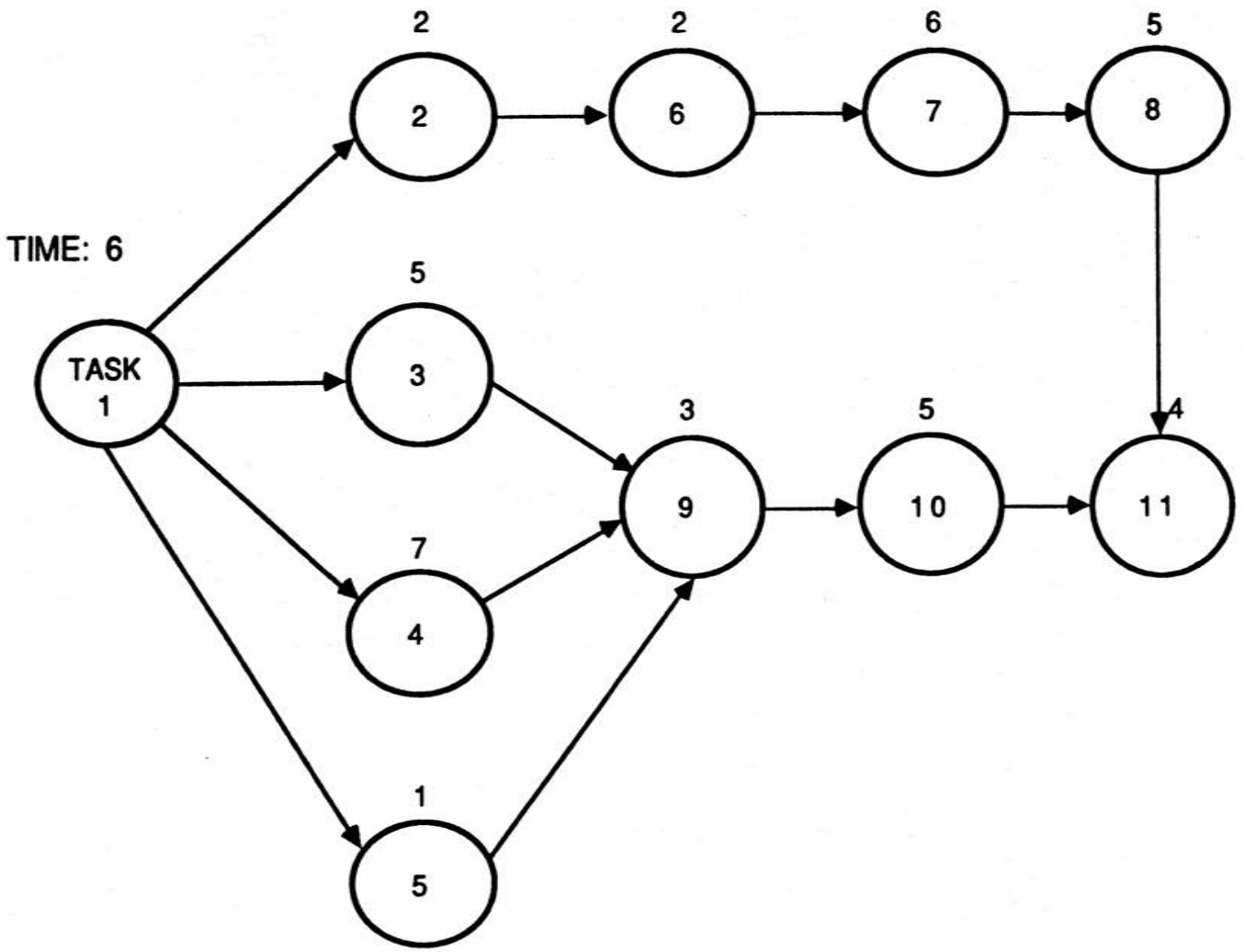

Figure 1: Assembly Line Balancing Example Problem due to Jackson (1956). The nodes represent the tasks to be assigned to stations and the arcs represent precedence relations. Each node has its associated task's processing time. The specified cycle time determines the work capacity of each station. There are 11 tasks. The total task time is 46 units.

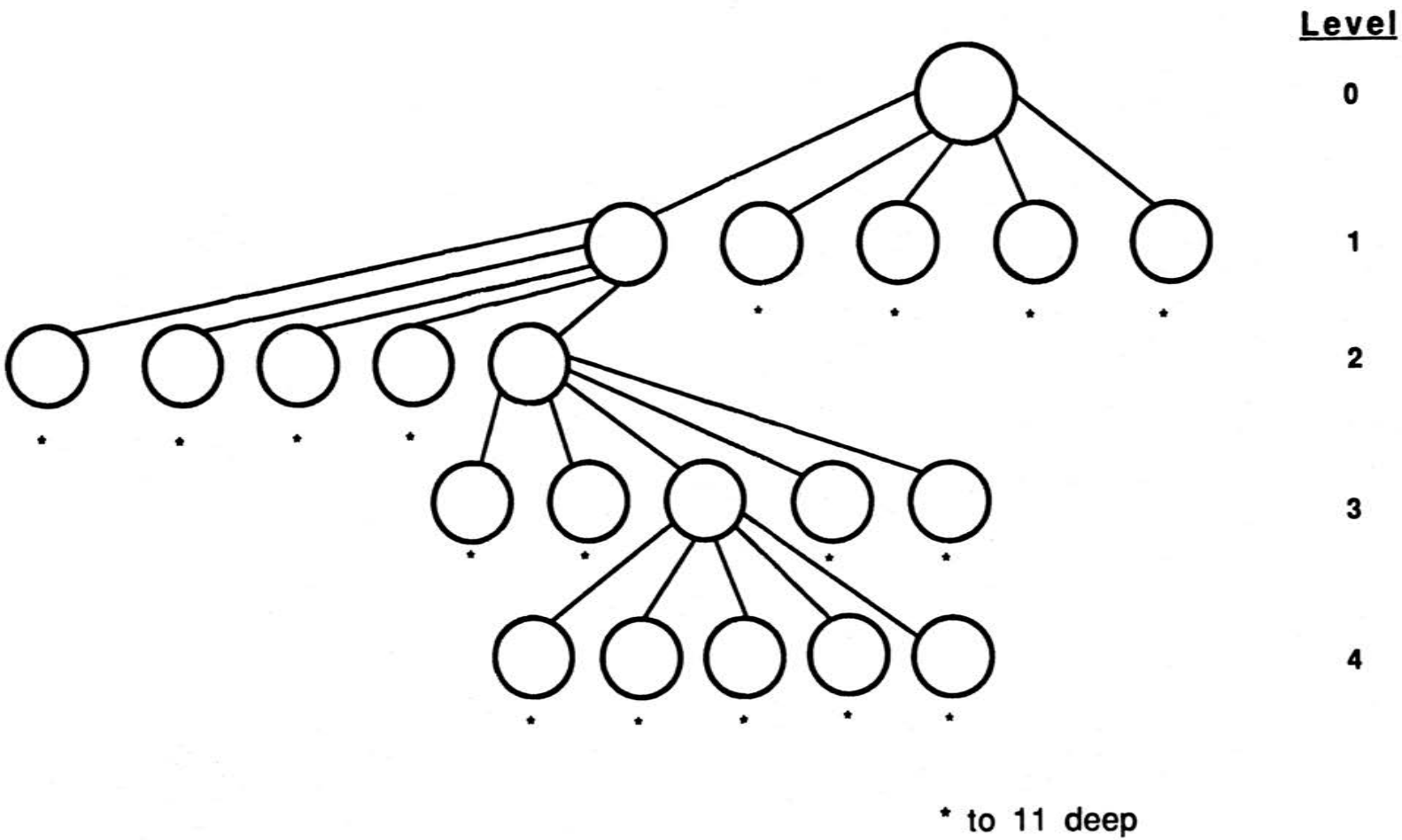

Figure 2: Complete Enumeration Tree for the Assembly Line Balancing Example Problem Shown in Figure 1 for a Cycle Time of 21 and a Maximum of 5 Stations. Each level of depth corresponds to a task, 1, ..., 11. The leftmost node (subproblem) corresponds to assigning the task at that level to the first station, the next node to the right corresponds to assignment to the second station, and so on to the fifth station. The minimum number of stations is 3. There are 61,035,156 nodes in this tree.

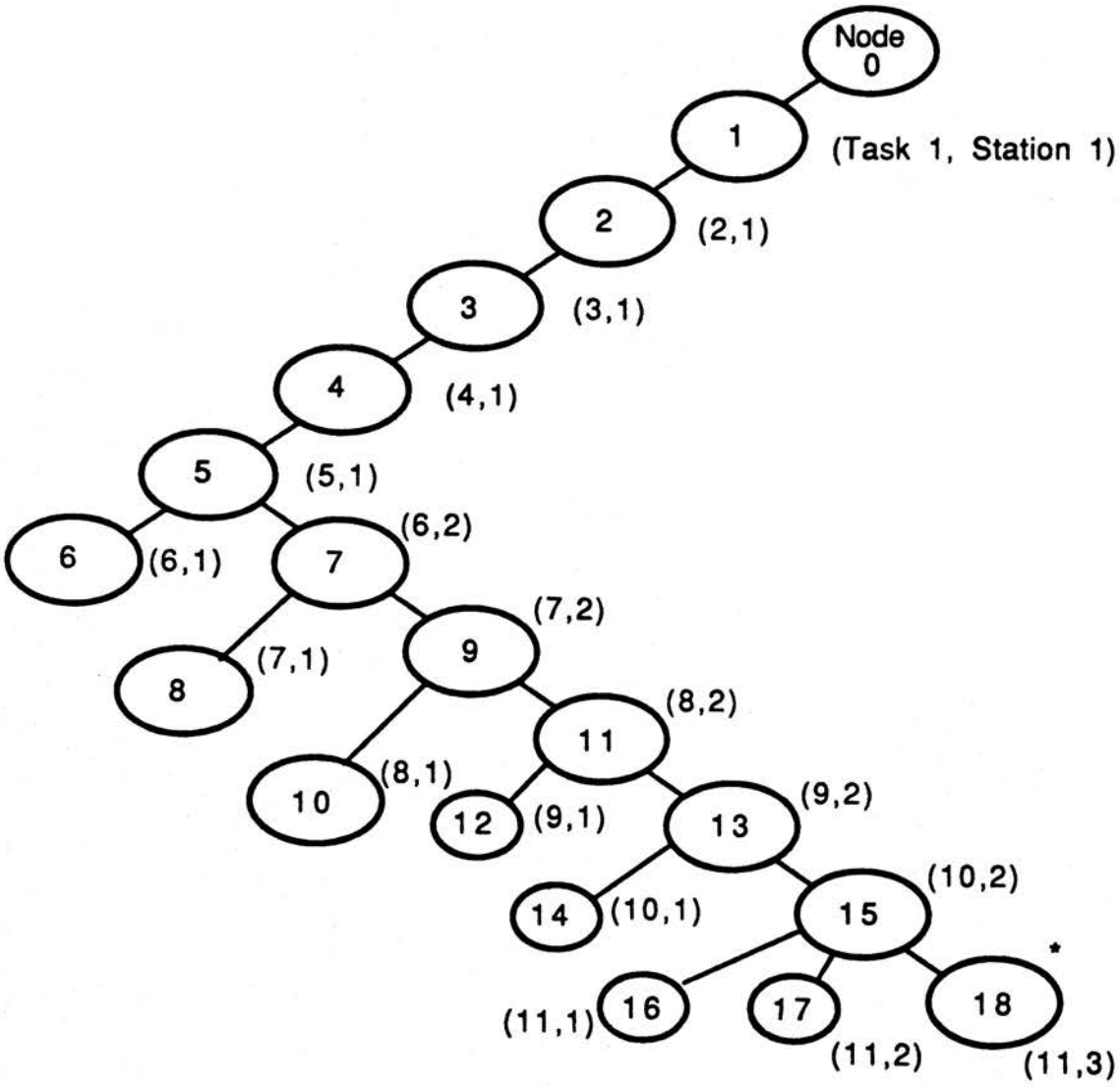

Figure 3: The Enumeration Tree for the Assembly Line Balancing Example Problem Shown in Figure 1 for a Cycle Time of 21 and a Maximum of 5 Stations. The problem was solved by the implicit enumeration algorithm. Inside each node is the node number. Outside each node in parenthesis are the (task number, station assignment). Large nodes are explicitly enumerated. Small nodes are implicitly enumerated. A heuristic has found an initial solution using 4 stations at node 0. The optimum is found at node 18 (indicated by a *) with the solution of tasks 1 - 5 in station 1, tasks 6 - 10 in station 2, and task 11 in station 3.

NOTE: The following is a partial list of papers that are currently available in the Edwin L. Cox School of Business Working Paper Series. When requesting a paper, please include the Working Paper number as well as the title and author(s), and enclose payment of $2.50 per copy made payable to SMU. A complete list is available upon request from:

Director, Research and Development
Edwin L. Cox School of Business
Southern Methodist University
Dallas, Texas 75275

86-084 "A Simple Algebraic Estimation Procedure for Innovation Diffusion Models of New Product Acceptance," by Vijay Mahajan and Subhash Sharma

86-091 "Some Probabilities Associated with the Ordering of Unknown Multinomial Cell Probabilities," by S. Y. Dennis

86-092 "The Incidence of Secured Lending: Evidence from the Small Business Community," by John D. Leeth, Jonathan A. Scott, and Terence C. Smith

86-093 "Workers' Compensation Insurance With Imperfect State Verification: The Long-run Impact on Injuries and Claims," by Thomas J. Kniesner and John D. Leeth

86-094 "Pricing and Diffusion of Primary and Contingent Products," by Vijay Mahajan and Eitan Muller

86-101 "Determination of Adopter Categories Using Innovation Diffusion Models," by Vijay Mahajan and Eitan Muller

86-111 "A Probabilistic Analysis of the Eigenvector Problem for Dominance Matrices of Unit Rank," by S. Y. Dennis

86-121 "Optimal Clustering: A Model and Method," by Gary Klein and Jay E. Aronson

86-122 "Question Effects on Information Processing in Advertising: A Comparative Model Approach," by Daniel J. Howard and Robert E. Burnkrant

86-123 "Question Effects on Information Processing: An Alternative Paradigm," by Daniel J. Howard

86-124 "Real Estate and the Tax Reform Act of 1986," by Patric H. Hendershott, James R. Follain, David C. Ling

87-011 "Criteria for Selecting Joint Venture Partners," by J. Michael Geringer

87-012 "A Model of the Joint Venture Partner Selection Process," by J. Michael Geringer

87-013 "Selection of Partners for International Joint Ventures," by J. Michael Geringer

87-021 "Metamorphosis in Strategic Market Planning," by Vijay Mahajan, P. "Rajan" Varadarajan, and Roger A. Kerin

87-022 "CEO Roles Across Cultures," by Ellen Jackofsky and John W. Slocum, Jr.

87-023 "Strategy Formulation Processes: Differences in Perceptions of Strength and Weaknesses Indicators and Environmental Uncertainty by Managerial Level," by R. Duane Ireland, Michael A. Hitt, Richard A. Bettis, and Deborah Auld De Porras

87-024 "Financial Returns and Strategic Interaction: The Case of Instant Photography," by Richard A. Bettis and David Weeks

87-031 "Interactive Multiobjective Optimization Under Uncertainty," by G. Klein, H. Moskowitz, and A. Ravindran

87-032 "Business Strategy, Staffing and Career Management Issues," by John W. Slocum, Jr. and William L. Cron

87-071 "Stages in the Evolution of Managerial Interpretation: A Study of Interpreting Key Organizational Events," by Lynn A. Isabella

87-072 "An Agent for Intelligent Model Management," by John I. C. Liu, David Y. Y. Yun, and Gary Klein

87-073 "Dynamics of the Career Plateauing Process," by Suzanne K. Stout, John W. Slocum, Jr., and William L. Cron

87-081 "Understanding the Real Estate Provisions of Tax Reform: Motivation and Impact," by James A. Follian, Patric H. Hendershott, and David Ling

87-082 "Home Ownership Rates of Married Couples: An Econometric Investigation," by Donald R. Haurin, Patric H. Hendershott, David C. Ling

87-091 "A Longitudinal Study of Climates," by Ellen F. Jackofsky and John W. Slocum, Jr.

87-101 "Hidden Messages in Corporate Relocation," by Lynn A. Isabella and Suzyn Ornstein

87-111 "Cultural Values and the CEO: Alluring Companions?" by Ellen F. Jackofsky, John W. Slocum, Jr., and Sara J. McQuaid

87-121 "Life Stage Versus Career Stage: A Comparative Test of the Theories of Levinson and Super," by Suzyn Ornstein, William L. Cron, John W. Slocum, Jr.

88-051 "Optimal Process Structuring," by Gary Klein, Jay E. Aronson, Philip O. Beck, and Benn R. Konsynski.

88-061 "The Solution of Computer Assisted Process Organization Problems with a Cluster Analysis Algorithm," by Jay E. Aronson and Gary Klein

88-062 "An Exact Algorithm for Variations of the Assembly Line Balancing Problem," by Jay E. Aronson and Gary Klein